Soundness and Completeness of an Axiomatic System for First-Order Logic

Asta Halkjær From

December 14, 2021

Abstract

This work is a formalization of the soundness and completeness of an axiomatic system for first-order logic. The proof system is based on System Q1 by Smullyan and the completeness proof follows his textbook "First-Order Logic" (Springer-Verlag 1968) [2]. The completeness proof is in the Henkin style [1] where a consistent set is extended to a maximal consistent set using Lindenbaum's construction and Henkin witnesses are added during the construction to ensure saturation as well. The resulting set is a Hintikka set which, by the model existence theorem, is satisfiable in the Herbrand universe.

Contents

theory *FOL-Axiomatic* **imports** *HOL−Library.Countable* **begin**

1 Syntax

datatype (*params-tm*: *'f*) *tm*
  = *Var nat* (‹#›)
  | *Fun 'f* ‹*'f tm list*› (‹†›)

abbreviation *Const* (‹⋆›) **where** ‹⋆*a* ≡ †*a* []›

datatype (*params-fm*: *'f*, *'p*) *fm*
  = *Falsity* (‹⊥›)
  | *Pre 'p* ‹*'f tm list*› (‹‡›)
  | *Imp* ‹(*'f*, *'p*) *fm*› ‹(*'f*, *'p*) *fm*› (**infixr** ‹⟶› *25*)
  | *Uni* ‹(*'f*, *'p*) *fm*› (‹∀›)

abbreviation *Neg* (‹¬ -› [*40*] *40*) **where** ‹¬ *p* ≡ *p* ⟶ ⊥›

term ‹∀(⊥ ⟶ ‡*''P''* [†*''f''* [#*0*]])›

2 Semantics

definition *shift* :: ‹(*nat* ⇒ *'a*) ⇒ *nat* ⇒ *'a* ⇒ *nat* ⇒ *'a*›
  (‹-⟨-:-⟩› [*90*, *0*, *0*] *91*) **where**
  ‹*E*⟨*n*:*x*⟩ = (λ*m*. *if m* < *n then E m else if m* = *n then x else E* (*m*−*1*))›

primrec *semantics-tm* (‹⦇-, -⦈›) **where**
  ‹⦇*E*, *F*⦈ (#*n*) = *E n*›
| ‹⦇*E*, *F*⦈ (†*f ts*) = *F f* (*map* ⦇*E*, *F*⦈ *ts*)›

primrec *semantics-fm* (‹⟦-, -, -⟧›) **where**
  ‹⟦-, -, -⟧ ⊥ = *False*›
| ‹⟦*E*, *F*, *G*⟧ (‡*P ts*) = *G P* (*map* ⦇*E*, *F*⦈ *ts*)›
| ‹⟦*E*, *F*, *G*⟧ (*p* ⟶ *q*) = (⟦*E*, *F*, *G*⟧ *p* ⟶ ⟦*E*, *F*, *G*⟧ *q*)›
| ‹⟦*E*, *F*, *G*⟧ (∀ *p*) = (∀ *x*. ⟦*E*⟨*0*:*x*⟩, *F*, *G*⟧ *p*)›

proposition ‹⟦*E*, *F*, *G*⟧ (∀ (‡*P* [# *0*]) ⟶ ‡*P* [⋆*a*])›
  by (*simp add*: *shift-def*)

3 Operations

3.1 Shift

lemma *shift-eq* [*simp*]: ‹*n* = *m* ⟹ (*E*⟨*n*:*x*⟩) *m* = *x*›
  by (*simp add*: *shift-def*)

lemma *shift-gt* [*simp*]: ‹$m < n \Longrightarrow (E\langle n{:}x\rangle)\ m = E\ m$›
by (*simp add*: *shift-def*)

lemma *shift-lt* [*simp*]: ‹$n < m \Longrightarrow (E\langle n{:}x\rangle)\ m = E\ (m{-}1)$›
by (*simp add*: *shift-def*)

lemma *shift-commute* [*simp*]: ‹$E\langle n{:}y\rangle\langle 0{:}x\rangle = E\langle 0{:}x\rangle\langle n{+}1{:}y\rangle$›
proof
fix *m*
show ‹$(E\langle n{:}y\rangle\langle 0{:}x\rangle)\ m = (E\langle 0{:}x\rangle\langle n{+}1{:}y\rangle)\ m$›
unfolding *shift-def* **by** (*cases m*) *simp-all*
qed

3.2 Parameters

abbreviation ‹*params S* ≡ $\bigcup p \in S.$ *params-fm p*›

lemma *upd-params-tm* [*simp*]: ‹$f \notin$ *params-tm t* $\Longrightarrow (\!|E,\ F(f := x)|\!)\ t = (\!|E,\ F|\!)\ t$›
by (*induct t*) (*auto cong*: *map-cong*)

lemma *upd-params-fm* [*simp*]: ‹$f \notin$ *params-fm p* $\Longrightarrow [\![E,\ F(f := x),\ G]\!]\ p = [\![E,\ F,\ G]\!]\ p$›
by (*induct p arbitrary*: *E*) (*auto cong*: *map-cong*)

lemma *finite-params-tm* [*simp*]: ‹*finite* (*params-tm t*)›
by (*induct t*) *simp-all*

lemma *finite-params-fm* [*simp*]: ‹*finite* (*params-fm p*)›
by (*induct p*) *simp-all*

3.3 Instantiation

primrec *lift-tm* (‹↑›) **where**
‹↑(#*n*) = #(*n*+1)›
| ‹↑(†*f ts*) = †*f* (*map* ↑ *ts*)›

primrec *inst-tm* (‹-'⟪-'/-'⟫› [*90*, *0*, *0*] *91*) **where**
‹(#*n*)⟪*s*/*m*⟫ = (*if n* < *m then* #*n else if n* = *m then s else* #(*n*−1))›
| ‹(†*f ts*)⟪*s*/*m*⟫ = †*f* (*map* (λ*t*. *t*⟪*s*/*m*⟫) *ts*)›

primrec *inst-fm* (‹-'⟨-'/-'⟩› [*90*, *0*, *0*] *91*) **where**
‹⊥⟨-/-⟩ = ⊥›
| ‹(‡*P ts*)⟨*s*/*m*⟩ = ‡*P* (*map* (λ*t*. *t*⟪*s*/*m*⟫) *ts*)›
| ‹(*p* ⟶ *q*)⟨*s*/*m*⟩ = (*p*⟨*s*/*m*⟩ ⟶ *q*⟨*s*/*m*⟩)›
| ‹(∀ *p*)⟨*s*/*m*⟩ = ∀ (*p*⟨↑*s*/*m*+1⟩)›

lemma *lift-lemma* [*simp*]: ‹$(\!|E\langle 0{:}x\rangle,\ F|\!)\ (\uparrow t) = (\!|E,\ F|\!)\ t$›
by (*induct t*) (*auto cong*: *map-cong*)

lemma *inst-tm-semantics* [*simp*]: ‹⦇*E*, *F*⦈ (*t*⟪*s*/*m*⟫) = ⦇*E*⟨*m*:⦇*E*, *F*⦈ *s*⟩, *F*⦈ *t*›
by (*induct t*) (*auto cong*: *map-cong*)

lemma *inst-fm-semantics* [*simp*]: ‹⟦*E*, *F*, *G*⟧ (*p*⟨*t*/*m*⟩) = ⟦*E*⟨*m*:⦇*E*, *F*⦈ *t*⟩, *F*, *G*⟧ *p*›
by (*induct p arbitrary*: *E m t*) (*auto cong*: *map-cong*)

3.4 Size

The built-in *size* is not invariant under substitution.

primrec *size-fm* **where**
‹*size-fm* ⊥ = *1*›
| ‹*size-fm* (‡- -) = *1*›
| ‹*size-fm* (*p* ⟶ *q*) = *1* + *size-fm p* + *size-fm q*›
| ‹*size-fm* (∀ *p*) = *1* + *size-fm p*›

lemma *size-inst-fm* [*simp*]:
‹*size-fm* (*p*⟨*t*/*m*⟩) = *size-fm p*›
by (*induct p arbitrary*: *m t*) *auto*

4 Propositional Semantics

primrec *boolean* **where**
‹*boolean* - - ⊥ = *False*›
| ‹*boolean G* - (‡*P ts*) = *G P ts*›
| ‹*boolean G A* (*p* ⟶ *q*) = (*boolean G A p* ⟶ *boolean G A q*)›
| ‹*boolean* - *A* (∀ *p*) = *A* (∀ *p*)›

abbreviation ‹*tautology p* ≡ ∀ *G A*. *boolean G A p*›

proposition ‹*tautology* (∀ (‡*P* [#*0*]) ⟶ ∀ (‡*P* [#*0*]))›
by *simp*

lemma *boolean-semantics*: ‹*boolean* (λ*a*. *G a* ∘ *map* ⦇*E*, *F*⦈) ⟦*E*, *F*, *G*⟧ = ⟦*E*, *F*, *G*⟧›
proof
fix *p*
show ‹*boolean* (λ*a*. *G a* ∘ *map* ⦇*E*, *F*⦈) ⟦*E*, *F*, *G*⟧ *p* = ⟦*E*, *F*, *G*⟧ *p*›
by (*induct p*) *simp-all*
qed

lemma *tautology*: ‹*tautology p* ⟹ ⟦*E*, *F*, *G*⟧ *p*›
using *boolean-semantics* **by** *metis*

proposition ‹∃ *p*. (∀ *E F G*. ⟦*E*, *F*, *G*⟧ *p*) ∧ ¬ *tautology p*›
by (*metis boolean.simps*(*4*) *fm.simps*(*36*) *semantics-fm.simps*(*1*,*3*,*4*))

5 Calculus

Adapted from System Q1 by Smullyan in First-Order Logic (1968)

inductive *Axiomatic* (‹⊢ -› [*20*] *20*) **where**
  TA: ‹*tautology p* ⟹ ⊢ *p*›
| *IA*: ‹⊢ ∀ *p* ⟶ *p*⟨*t*/*0*⟩›
| *MP*: ‹⊢ *p* ⟶ *q* ⟹ ⊢ *p* ⟹ ⊢ *q*›
| *GR*: ‹⊢ *q* ⟶ *p*⟨⋆*a*/*0*⟩ ⟹ *a* ∉ *params* {*p*, *q*} ⟹ ⊢ *q* ⟶ ∀ *p*›

lemmas
  TA[*simp*]
  MP[*trans*, *dest*]
  GR[*intro*]

We simulate assumptions on the lhs of ⊢ with a chain of implications on the rhs.

primrec *imply* (**infixr** ‹⇝› *26*) **where**
  ‹([] ⇝ *q*) = *q*›
| ‹(*p* # *ps* ⇝ *q*) = (*p* ⟶ *ps* ⇝ *q*)›

abbreviation *Axiomatic-assms* (‹- ⊢ -› [*20*, *20*] *20*) **where**
  ‹*ps* ⊢ *q* ≡ ⊢ *ps* ⇝ *q*›

6 Soundness

theorem *soundness*: ‹⊢ *p* ⟹ ⟦*E*, *F*, *G*⟧ *p*›
proof (*induct p arbitrary*: *F rule*: *Axiomatic.induct*)
  case (*GR q p a*)
  moreover from *this*
  have ‹⟦*E*, *F*(*a* := *x*), *G*⟧ (*q* ⟶ *p*⟨⋆*a*/*0*⟩)› **for** *x*
    by *blast*
  ultimately show *?case*
    by *fastforce*
qed (*auto simp*: *tautology*)

corollary ‹¬ (⊢ ⊥)›
  using *soundness* **by** *fastforce*

7 Derived Rules

lemma *AS*: ‹⊢ (*p* ⟶ *q* ⟶ *r*) ⟶ (*p* ⟶ *q*) ⟶ *p* ⟶ *r*›
  by *auto*

lemma *AK*: ‹⊢ *q* ⟶ *p* ⟶ *q*›
  by *auto*

lemma *Neg*: ‹⊢ ¬ ¬ *p* ⟶ *p*›

by *auto*

lemma *contraposition*:
‹$\vdash (p \longrightarrow q) \longrightarrow \neg\ q \longrightarrow \neg\ p$›
‹$\vdash (\neg\ q \longrightarrow \neg\ p) \longrightarrow p \longrightarrow q$›
by (*auto intro*: *TA*)

lemma *GR'*: ‹$\vdash \neg\ p\langle\star a/0\rangle \longrightarrow q \Longrightarrow a \notin params\ \{p, q\} \Longrightarrow \vdash \neg\ \forall\, p \longrightarrow q$›
proof −
assume ∗: ‹$\vdash \neg\ p\langle\star a/0\rangle \longrightarrow q$› **and** *a*: ‹$a \notin params\ \{p, q\}$›
have ‹$\vdash \neg\ q \longrightarrow \neg\ \neg\ p\langle\star a/0\rangle$›
using ∗ *contraposition*(*1*) **by** *fast*
then have ‹$\vdash \neg\ q \longrightarrow p\langle\star a/0\rangle$›
by (*meson AK AS MP Neg*)
then have ‹$\vdash \neg\ q \longrightarrow \forall\, p$›
using *a* **by** *auto*
then have ‹$\vdash \neg\ \forall\, p \longrightarrow \neg\ \neg\ q$›
using *contraposition*(*1*) **by** *fast*
then show *?thesis*
by (*meson AK AS MP Neg*)
qed

lemma *Imp3*: ‹$\vdash (p \longrightarrow q \longrightarrow r) \longrightarrow ((s \longrightarrow p) \longrightarrow (s \longrightarrow q) \longrightarrow s \longrightarrow r)$›
by *auto*

lemma *imply-ImpE*: ‹$\vdash ps \rightsquigarrow p \longrightarrow ps \rightsquigarrow (p \longrightarrow q) \longrightarrow ps \rightsquigarrow q$›
by (*induct ps*) (*auto intro*: *Imp3 MP*)

lemma *MP'* [*trans*, *dest*]: ‹$ps \vdash p \longrightarrow q \Longrightarrow ps \vdash p \Longrightarrow ps \vdash q$›
using *imply-ImpE* **by** *fast*

lemma *imply-Cons* [*intro*]: ‹$ps \vdash q \Longrightarrow p \#\ ps \vdash q$›
by (*auto intro*: *MP AK*)

lemma *imply-head* [*intro*]: ‹$p \#\ ps \vdash p$›
proof (*induct ps*)
case (*Cons q ps*)
then show *?case*
by (*metis AK MP' imply.simps*(*1*−*2*))
qed *auto*

lemma *imply-lift-Imp* [*simp*]:
assumes ‹$\vdash p \longrightarrow q$›
shows ‹$\vdash p \longrightarrow ps \rightsquigarrow q$›
using *assms MP MP' imply-head* **by** (*metis imply.simps*(*2*))

lemma *add-imply* [*simp*]: ‹$\vdash q \Longrightarrow ps \vdash q$›
using *MP imply-head* **by** (*auto simp del*: *TA*)

lemma *imply-mem* [*simp*]: ‹$p \in set\ ps \Longrightarrow ps \vdash p$›
proof (*induct ps*)
  case (*Cons q ps*)
  then show *?case*
    by (*metis imply-Cons imply-head set-ConsD*)
qed *simp*

lemma *deduct1*: ‹$ps \vdash p \longrightarrow q \Longrightarrow p \# ps \vdash q$›
  by (*meson MP′ imply-Cons imply-head*)

lemma *imply-append* [*iff*]: ‹$(ps @ qs \rightsquigarrow r) = (ps \rightsquigarrow qs \rightsquigarrow r)$›
  by (*induct ps*) *simp-all*

lemma *imply-swap-append*: ‹$ps @ qs \vdash r \Longrightarrow qs @ ps \vdash r$›
proof (*induct qs arbitrary*: *ps*)
  case (*Cons q qs*)
  then show *?case*
    by (*metis deduct1 imply.simps*(*2*) *imply-append*)
qed *simp*

lemma *deduct2*: ‹$p \# ps \vdash q \Longrightarrow ps \vdash p \longrightarrow q$›
  by (*metis imply.simps*(*1−2*) *imply-append imply-swap-append*)

lemmas *deduct* [*iff*] = *deduct1 deduct2*

lemma *cut* [*trans*, *dest*]: ‹$p \# ps \vdash r \Longrightarrow q \# ps \vdash p \Longrightarrow q \# ps \vdash r$›
  by (*meson MP′ deduct*(*2*) *imply-Cons*)

lemma *Boole*: ‹$(\neg p) \# ps \vdash \bot \Longrightarrow ps \vdash p$›
  by (*meson MP′ Neg add-imply deduct*(*2*))

lemma *imply-weaken*: ‹$ps \vdash q \Longrightarrow set\ ps \subseteq set\ ps' \Longrightarrow ps' \vdash q$›
proof (*induct ps arbitrary*: *q*)
  case (*Cons p ps*)
  then show *?case*
    by (*metis MP′ deduct*(*2*) *imply-mem insert-subset list.simps*(*15*))
qed *simp*

8 Consistent

definition ‹*consistent* $S \equiv \nexists S'.\ set\ S' \subseteq S \wedge (S' \vdash \bot)$›

lemma *UN-finite-bound*:
  assumes ‹*finite A*› **and** ‹$A \subseteq (\bigcup n.\ f\ n)$›
  shows ‹$\exists m :: nat.\ A \subseteq (\bigcup n \leq m.\ f\ n)$›
  using *assms*
proof (*induct rule*: *finite-induct*)
  case (*insert x A*)
  then obtain *m* **where** ‹$A \subseteq (\bigcup n \leq m.\ f\ n)$›

by *fast*
then have ‹$A \subseteq (\bigcup n \leq (m + k).\ f\ n)$› **for** k
by *fastforce*
moreover obtain m' **where** ‹$x \in f\ m'$›
using *insert*(*4*) **by** *blast*
ultimately have ‹$\{x\} \cup A \subseteq (\bigcup n \leq m + m'.\ f\ n)$›
by *auto*
then show *?case*
by *blast*
qed *simp*

lemma *split-list*:
assumes ‹$x \in$ *set* A›
shows ‹*set* $(x \#$ *removeAll* $x\ A) =$ *set* $A \wedge x \notin$ *set* (*removeAll* $x\ A)$›
using *assms* **by** *auto*

lemma *imply-params-fm*: ‹*params-fm* $(ps \rightsquigarrow q) =$ *params-fm* $q \cup (\bigcup p \in$ *set* $ps.$ *params-fm* $p)$›
by (*induct* ps) *auto*

lemma *inconsistent-fm*:
assumes ‹*consistent* S› **and** ‹$\neg$ *consistent* $(\{p\} \cup S)$›
obtains S' **where** ‹*set* $S' \subseteq S$› **and** ‹$p \#\ S' \vdash \bot$›
proof −
obtain S' **where** S': ‹*set* $S' \subseteq \{p\} \cup S$› ‹$p \in$ *set* S'› ‹$S' \vdash \bot$›
using *assms* **unfolding** *consistent-def* **by** *blast*
then obtain S'' **where** S'': ‹*set* $(p \#\ S'') =$ *set* S'› ‹$p \notin$ *set* S''›
using *split-list* **by** *metis*
then have ‹$p \#\ S'' \vdash \bot$›
using ‹$S' \vdash \bot$› *imply-weaken* **by** *blast*
then show *?thesis*
using *that* S'' $S'(1)$
by (*metis Diff-insert-absorb Diff-subset-conv list.simps*(*15*))
qed

lemma *consistent-add-witness*:
assumes ‹*consistent* S› **and** ‹$(\neg \forall p) \in S$› **and** ‹$a \notin$ *params* S›
shows ‹*consistent* $(\{\neg\ p\langle\star a/0\rangle\} \cup S)$›
unfolding *consistent-def*
proof
assume ‹$\exists S'.$ *set* $S' \subseteq \{\neg\ p\langle\star a/0\rangle\} \cup S \wedge (S' \vdash \bot)$›
then obtain S' **where** ‹*set* $S' \subseteq S$› **and** ‹$(\neg\ p\langle\star a/0\rangle) \#\ S' \vdash \bot$›
using *assms inconsistent-fm* **unfolding** *consistent-def* **by** *metis*
then have ‹$\vdash \neg\ p\langle\star a/0\rangle \longrightarrow S' \rightsquigarrow \bot$›
by *simp*
moreover have ‹$a \notin$ *params-fm* p›
using *assms*(*2*−*3*) **by** *auto*
moreover have ‹$\forall p \in$ *set* $S'.\ a \notin$ *params-fm* p›
using ‹*set* $S' \subseteq S$› *assms*(*3*) **by** *auto*

then have ‹$a \notin$ *params-fm* $(S' \rightsquigarrow \bot)$›
by (*simp add*: *imply-params-fm*)
ultimately have ‹$\vdash \neg \forall\, p \longrightarrow S' \rightsquigarrow \bot$›
using *GR'* **by** *fast*
then have ‹$(\neg \forall\, p) \# S' \vdash \bot$›
by *simp*
moreover have ‹*set* $((\neg \forall\, p) \# S') \subseteq S$›
using ‹*set* $S' \subseteq S$› *assms*(*2*) **by** *simp*
ultimately show *False*
using *assms*(*1*) **unfolding** *consistent-def* **by** *blast*
qed

lemma *consistent-add-instance*:
assumes ‹*consistent* S› **and** ‹$\forall\, p \in S$›
shows ‹*consistent* $(\{p\langle t/0\rangle\} \cup S)$›
unfolding *consistent-def*
proof
assume ‹$\exists\, S'$. *set* $S' \subseteq \{p\langle t/0\rangle\} \cup S \wedge (S' \vdash \bot)$›
then obtain S' **where** ‹*set* $S' \subseteq S$› **and** ‹$p\langle t/0\rangle \# S' \vdash \bot$›
using *assms inconsistent-fm* **unfolding** *consistent-def* **by** *blast*
moreover have ‹$\vdash \forall\, p \longrightarrow p\langle t/0\rangle$›
using *IA* **by** *blast*
ultimately have ‹$\forall\, p \# S' \vdash \bot$›
by (*meson add-imply cut deduct*(*1*))
moreover have ‹*set* $((\forall\, p) \# S') \subseteq S$›
using ‹*set* $S' \subseteq S$› *assms*(*2*) **by** *simp*
ultimately show *False*
using *assms*(*1*) **unfolding** *consistent-def* **by** *blast*
qed

9 Extension

fun *witness* **where**
‹*witness used* $(\neg \forall\, p) = \{\neg\, p\langle \star(\mathit{SOME}\ a.\ a \notin \mathit{used})/0\rangle\}$›
| ‹*witness* - - = $\{\}$›

primrec *extend* **where**
‹*extend* $S\ f\ 0 = S$›
| ‹*extend* $S\ f\ (\mathit{Suc}\ n) =$
(*let* Sn = *extend* $S\ f\ n$ *in*
if consistent $(\{f\ n\} \cup Sn)$
then witness (*params* $(\{f\ n\} \cup Sn)$) $(f\ n) \cup \{f\ n\} \cup Sn$
else Sn)›

definition ‹*Extend* $S\ f \equiv \bigcup n.$ *extend* $S\ f\ n$›

lemma *Extend-subset*: ‹$S \subseteq$ *Extend* $S\ f$›
unfolding *Extend-def* **by** (*metis Union-upper extend.simps*(*1*) *range-eqI*)

lemma *extend-bound*: ‹$(\bigcup n \leq m.\ extend\ S\ f\ n) = extend\ S\ f\ m$›
by (*induct m*) (*simp-all add*: *atMost-Suc Let-def*)

lemma *finite-params-witness* [*simp*]: ‹*finite* (*params* (*witness used p*))›
by (*induct used p rule*: *witness.induct*) *simp-all*

lemma *finite-params-extend* [*simp*]: ‹*finite* (*params S*) $\Longrightarrow$ *finite* (*params* (*extend S f n*))›
by (*induct n*) (*simp-all add*: *Let-def*)

lemma *consistent-witness*:
fixes *p* :: ‹('*f*, '*p*) *fm*›
assumes ‹*consistent S*› **and** ‹$p \in S$› **and** ‹$params\ S \subseteq used$›
and ‹*finite used*› **and** ‹*infinite* (*UNIV* :: '*f set*)›
shows ‹*consistent* (*witness used p* $\cup$ *S*)›
using *assms*
proof (*induct used p rule*: *witness.induct*)
case (*1 used p*)
moreover have ‹$\exists a.\ a \notin used$›
using *1*(*4*−) *ex-new-if-finite* **by** *blast*
ultimately obtain *a* **where** *a*: ‹$witness\ used\ (\neg\ \forall\ p) = \{\neg\ p\langle\star a/0\rangle\}$› **and** ‹$a \notin used$›
by (*metis someI-ex witness.simps*(*1*))
then have ‹$a \notin params\ S$›
using *1*(*3*) **by** *fast*
then show *?case*
using *1*(*1*−*2*) *a*(*1*) *consistent-add-witness* **by** *metis*
qed (*auto simp*: *assms*)

lemma *consistent-extend*:
fixes *f* :: ‹*nat* ⇒ ('*f*, '*p*) *fm*›
assumes ‹*consistent S*› **and** ‹*finite* (*params S*)›
and ‹*infinite* (*UNIV* :: '*f set*)›
shows ‹*consistent* (*extend S f n*)›
using *assms*
proof (*induct n*)
case (*Suc n*)
then show *?case*
using *consistent-witness*[**where** *S*=‹$\{f\ n\} \cup$ -›] **by** (*auto simp*: *Let-def*)
qed *simp*

lemma *consistent-Extend*:
fixes *f* :: ‹*nat* ⇒ ('*f*, '*p*) *fm*›
assumes ‹*consistent S*› **and** ‹*finite* (*params S*)›
and ‹*infinite* (*UNIV* :: '*f set*)›
shows ‹*consistent* (*Extend S f*)›
proof (*rule ccontr*)
assume ‹$\neg$ *consistent* (*Extend S f*)›
then obtain *S'* **where** ‹$S' \vdash \bot$› **and** ‹$set\ S' \subseteq Extend\ S\ f$›

unfolding *consistent-def* **by** *blast*
then obtain m **where** ‹*set* $S' \subseteq (\bigcup n \leq m.\ extend\ S\ f\ n)$›
unfolding *Extend-def* **using** *UN-finite-bound* **by** (*metis List.finite-set*)
then have ‹*set* $S' \subseteq extend\ S\ f\ m$›
using *extend-bound* **by** *blast*
moreover have ‹*consistent* (*extend* $S\ f\ m$)›
using *assms consistent-extend* **by** *blast*
ultimately show *False*
unfolding *consistent-def* **using** ‹$S' \vdash \bot$› **by** *blast*
qed

10 Maximal

definition ‹*maximal* $S \equiv \forall p.\ p \notin S \longrightarrow \neg\ consistent\ (\{p\} \cup S)$›

lemma *maximal-exactly-one*:
assumes ‹*consistent* S› **and** ‹*maximal* S›
shows ‹$p \in S \longleftrightarrow (\neg\ p) \notin S$›
proof
assume ‹$p \in S$›
show ‹$(\neg\ p) \notin S$›
proof
assume ‹$(\neg\ p) \in S$›
then have ‹*set* $[p, \neg\ p] \subseteq S$›
using ‹$p \in S$› **by** *simp*
moreover have ‹$[p, \neg\ p] \vdash \bot$›
by *blast*
ultimately show *False*
using ‹*consistent* S› **unfolding** *consistent-def* **by** *blast*
qed
next
assume ‹$(\neg\ p) \notin S$›
then have ‹$\neg$ *consistent* $(\{\neg\ p\} \cup S)$›
using ‹*maximal* S› **unfolding** *maximal-def* **by** *blast*
then obtain S' **where** ‹*set* $S' \subseteq S$› ‹$(\neg\ p)\ \#\ S' \vdash \bot$›
using ‹*consistent* S› *inconsistent-fm* **by** *blast*
then have ‹$S' \vdash p$›
using *Boole* **by** *blast*
have ‹*consistent* $(\{p\} \cup S)$›
unfolding *consistent-def*
proof
assume ‹$\exists S'.$ *set* $S' \subseteq \{p\} \cup S \wedge (S' \vdash \bot)$›
then obtain S'' **where** ‹*set* $S'' \subseteq S$› **and** ‹$p\ \#\ S'' \vdash \bot$›
using *assms inconsistent-fm* **unfolding** *consistent-def* **by** *blast*
then have ‹$S' @ S'' \vdash \bot$›
using ‹$S' \vdash p$› **by** (*metis MP' add-imply imply.simps*(*2*) *imply-append*)
moreover have ‹*set* $(S' @ S'') \subseteq S$›
using ‹*set* $S' \subseteq S$› ‹*set* $S'' \subseteq S$› **by** *simp*
ultimately show *False*

using ‹*consistent S*› **unfolding** *consistent-def* **by** *blast*
qed
then show ‹$p \in S$›
using ‹*maximal S*› **unfolding** *maximal-def* **by** *blast*
qed

lemma *maximal-Extend*:
assumes ‹*surj f*›
shows ‹*maximal* (*Extend S f*)›
proof (*rule ccontr*)
assume ‹$\neg$ *maximal* (*Extend S f*)›
then obtain p **where**
‹$p \notin$ *Extend S f*› **and** ‹*consistent* ($\{p\} \cup$ *Extend S f*)›
unfolding *maximal-def* **using** *assms consistent-Extend* **by** *blast*
obtain k **where** k: ‹$f\ k = p$›
using ‹*surj f*› **unfolding** *surj-def* **by** *metis*
then have ‹$p \notin$ *extend S f* (*Suc k*)›
using ‹$p \notin$ *Extend S f*› **unfolding** *Extend-def* **by** *blast*
then have ‹$\neg$ *consistent* ($\{p\} \cup$ *extend S f k*)›
using k **by** (*auto simp*: *Let-def*)
moreover have ‹$\{p\} \cup$ *extend S f k* $\subseteq \{p\} \cup$ *Extend S f*›
unfolding *Extend-def* **by** *blast*
ultimately have ‹$\neg$ *consistent* ($\{p\} \cup$ *Extend S f*)›
unfolding *consistent-def* **by** *auto*
then show *False*
using ‹*consistent* ($\{p\} \cup$ *Extend S f*)› **by** *blast*
qed

11 Saturation

definition ‹*saturated S* $\equiv \forall p.\ (\neg \forall p) \in S \longrightarrow (\exists a.\ (\neg\ p\langle\star a/0\rangle) \in S)$›

lemma *saturated-Extend*:
assumes ‹*consistent* (*Extend S f*)› **and** ‹*surj f*›
shows ‹*saturated* (*Extend S f*)›
proof (*rule ccontr*)
assume ‹$\neg$ *saturated* (*Extend S f*)›
then obtain p **where** p: ‹$(\neg \forall p) \in$ *Extend S f*› ‹$\nexists a.\ (\neg\ p\langle\star a/0\rangle) \in$ *Extend S f*›
unfolding *saturated-def* **by** *blast*
obtain k **where** k: ‹$f\ k = (\neg \forall p)$›
using ‹*surj f*› **unfolding** *surj-def* **by** *metis*

have ‹*extend S f k* $\subseteq$ *Extend S f*›
unfolding *Extend-def* **by** *auto*
then have ‹*consistent* ($\{\neg \forall p\} \cup$ *extend S f k*)›
using *assms*(*1*) *p*(*1*) **unfolding** *consistent-def* **by** *blast*
then have ‹$\exists a.$ *extend S f* (*Suc k*) $= \{\neg\ p\langle\star a/0\rangle\} \cup \{\neg \forall p\} \cup$ *extend S f k*›
using k **by** (*auto simp*: *Let-def*)

moreover have ‹*extend S f* (*Suc k*) ⊆ *Extend S f*›
unfolding *Extend-def* **by** *blast*
ultimately show *False*
using *p*(*2*) **by** *blast*
qed

12 Hintikka

locale *Hintikka* =
fixes *H* :: ‹('*f*, '*p*) *fm set*›
assumes
NoFalsity: ‹⊥ ∉ *H*› **and**
ImpP: ‹(*p* ⟶ *q*) ∈ *H* ⟹ *p* ∉ *H* ∨ *q* ∈ *H*› **and**
ImpN: ‹(*p* ⟶ *q*) ∉ *H* ⟹ *p* ∈ *H* ∧ *q* ∉ *H*› **and**
UniP: ‹∀ *p* ∈ *H* ⟹ ∀ *t*. *p*⟨*t*/*0*⟩ ∈ *H*› **and**
UniN: ‹∀ *p* ∉ *H* ⟹ ∃ *a*. *p*⟨⋆*a*/*0*⟩ ∉ *H*›

12.1 Model Existence

abbreviation *hmodel* (‹⟦-⟧›) **where** ‹⟦*H*⟧ ≡ ⟦#, †, λ*P ts*. *Pre P ts* ∈ *H*⟧›

lemma *semantics-tm-id* [*simp*]:
‹(|#, †|) *t* = *t*›
by (*induct t*) (*auto cong*: *map-cong*)

lemma *semantics-tm-id-map* [*simp*]: ‹*map* (|#, †|) *ts* = *ts*›
by (*auto cong*: *map-cong*)

theorem *Hintikka-model*:
assumes ‹*Hintikka H*›
shows ‹*p* ∈ *H* ⟷ ⟦*H*⟧ *p*›
proof (*induct p rule*: *wf-induct*[**where** *r*=‹*measure size-fm*›])
case *1*
then show *?case* **..**
next
case (*2 x*)
show ‹*x* ∈ *H* ⟷ ⟦*H*⟧ *x*›
proof (*cases x*; *safe*)
case *Falsity*
assume ‹⊥ ∈ *H*›
then have *False*
using *assms Hintikka.NoFalsity* **by** *fast*
then show ‹⟦*H*⟧ ⊥› **..**
next
case *Falsity*
assume ‹⟦*H*⟧ ⊥›
then have *False*
by *simp*
then show ‹⊥ ∈ *H*› **..**

```
  next
    case (Pre P ts)
    assume ‹‡P ts ∈ H›
    then show ‹⟦H⟧ (‡P ts)›
      by simp
  next
    case (Pre P ts)
    assume ‹⟦H⟧ (‡P ts)›
    then show ‹‡P ts ∈ H›
      by simp
  next
    case (Imp p q)
    assume ‹(p ⟶ q) ∈ H›
    then have ‹p ∉ H ∨ q ∈ H›
      using assms Hintikka.ImpP by blast
    then have ‹¬ ⟦H⟧ p ∨ ⟦H⟧ q›
      using 2 Imp by simp
    then show ‹⟦H⟧ (p ⟶ q)›
      by simp
  next
    case (Imp p q)
    assume ‹⟦H⟧ (p ⟶ q)›
    then have ‹¬ ⟦H⟧ p ∨ ⟦H⟧ q›
      by simp
    then have ‹p ∉ H ∨ q ∈ H›
      using 2 Imp by simp
    then show ‹(p ⟶ q) ∈ H›
      using assms Hintikka.ImpN by blast
  next
    case (Uni p)
    assume ‹∀ p ∈ H›
    then have ‹∀ t. p⟨t/0⟩ ∈ H›
      using assms Hintikka.UniP by metis
    then have ‹∀ t. ⟦H⟧ (p⟨t/0⟩)›
      using 2 Uni by simp
    then show ‹⟦H⟧ (∀ p)›
      by simp
  next
    case (Uni p)
    assume ‹⟦H⟧ (∀ p)›
    then have ‹∀ t. ⟦H⟧ (p⟨t/0⟩)›
      by simp
    then have ‹∀ t. p⟨t/0⟩ ∈ H›
      using 2 Uni by simp
    then show ‹∀ p ∈ H›
      using assms Hintikka.UniN by blast
  qed
qed
```

12.2 Maximal Consistent Sets are Hintikka Sets

lemma *inconsistent-head*:
assumes ‹*consistent S*› **and** ‹*maximal S*› **and** ‹$p \notin S$›
obtains S' **where** ‹*set* $S' \subseteq S$› **and** ‹$p \# S' \vdash \bot$›
using *assms inconsistent-fm* **unfolding** *consistent-def maximal-def* **by** *metis*

lemma *inconsistent-parts* [*simp*]:
assumes ‹$ps \vdash \bot$› **and** ‹*set ps* $\subseteq S$›
shows ‹$\neg$ *consistent S*›
using *assms* **unfolding** *consistent-def* **by** *blast*

lemma *Hintikka-Extend*:
fixes H :: ‹('*f*, '*p*) *fm set*›
assumes ‹*consistent H*› **and** ‹*maximal H*› **and** ‹*saturated H*›
and ‹*infinite* (*UNIV* :: '*f set*)›
shows ‹*Hintikka H*›
proof
show ‹$\bot \notin H$›
proof
assume ‹$\bot \in H$›
moreover have ‹$[\bot] \vdash \bot$›
by *blast*
ultimately have ‹$\neg$ *consistent H*›
using *inconsistent-parts*[**where** *ps*=‹$[\bot]$›] **by** *simp*
then show *False*
using ‹*consistent H*› **..**
qed
next
fix *p q*
assume $*$: ‹$(p \longrightarrow q) \in H$›
show ‹$p \notin H \vee q \in H$›
proof *safe*
assume ‹$q \notin H$›
then obtain Hq' **where** Hq': ‹$q \# Hq' \vdash \bot$› ‹*set* $Hq' \subseteq H$›
using *assms inconsistent-head* **by** *metis*

assume ‹$p \in H$›
then have ‹$(\neg\ p) \notin H$›
using *assms maximal-exactly-one* **by** *blast*
then obtain Hp' **where** Hp': ‹$(\neg\ p) \# Hp' \vdash \bot$› ‹*set* $Hp' \subseteq H$›
using *assms inconsistent-head* **by** *metis*

let *?H'* = ‹Hp' @ Hq'›
have H': ‹*set ?H'* = *set* $Hp' \cup$ *set* Hq'›
by *simp*
then have ‹*set* $Hp' \subseteq$ *set ?H'*› **and** ‹*set* $Hq' \subseteq$ *set ?H'*›
by *blast+*
then have ‹$(\neg\ p) \# ?H' \vdash \bot$› **and** ‹$q \# ?H' \vdash \bot$›
using $Hp'(1)$ $Hq'(1)$ *deduct imply-weaken* **by** *metis+*

```
    then have ‹(p ⟶ q) # ?H' ⊢ ⊥›
      using Boole imply-Cons imply-head MP' cut by metis
    moreover have ‹set ((p ⟶ q) # ?H') ⊆ H›
      using ‹q ∉ H› ∗(1) H' Hp'(2) Hq'(2) by auto
    ultimately show False
      using assms unfolding consistent-def by blast
  qed
next
  fix p q
  assume ∗: ‹(p ⟶ q) ∉ H›
  show ‹p ∈ H ∧ q ∉ H›
  proof (safe, rule ccontr)
    assume ‹p ∉ H›
    then obtain H' where S': ‹p # H' ⊢ ⊥› ‹set H' ⊆ H›
      using assms inconsistent-head by metis
    moreover have ‹(¬ (p ⟶ q)) # H' ⊢ p›
      by auto
    ultimately have ‹(¬ (p ⟶ q)) # H' ⊢ ⊥›
      by blast
    moreover have ‹set ((¬ (p ⟶ q)) # H') ⊆ H›
      using ∗(1) S'(2) assms maximal-exactly-one by auto
    ultimately show False
      using assms unfolding consistent-def by blast
  next
    assume ‹q ∈ H›
    then have ‹(¬ q) ∉ H›
      using assms maximal-exactly-one by blast
    then obtain H' where H': ‹(¬ q) # H' ⊢ ⊥› ‹set H' ⊆ H›
      using assms inconsistent-head by metis
    moreover have ‹(¬ (p ⟶ q)) # H' ⊢ ¬ q›
      by auto
    ultimately have ‹(¬ (p ⟶ q)) # H' ⊢ ⊥›
      by blast
    moreover have ‹set ((¬ (p ⟶ q)) # H') ⊆ H›
      using ∗(1) H'(2) assms maximal-exactly-one by auto
    ultimately show False
      using assms unfolding consistent-def by blast
  qed
next
  fix p
  assume ‹∀ p ∈ H›
  then show ‹∀ t. p⟨t/0⟩ ∈ H›
    using assms consistent-add-instance unfolding maximal-def by blast
next
  fix p
  assume ‹∀ p ∉ H›
  then show ‹∃ a. p⟨⋆a/0⟩ ∉ H›
    using assms maximal-exactly-one unfolding saturated-def by fast
qed
```

13 Countable Formulas

instance *tm* :: (*countable*) *countable*
by *countable-datatype*

instance *fm* :: (*countable*, *countable*) *countable*
by *countable-datatype*

14 Completeness

lemma *imply-completeness*:
fixes p :: ‹('*f* :: *countable*, '*p* :: *countable*) *fm*›
assumes ‹∀(*E* :: - ⇒ '*f tm*) *F G*. *Ball X* ⟦*E*, *F*, *G*⟧ ⟶ ⟦*E*, *F*, *G*⟧ *p*›
and ‹*finite* (*params X*)› **and** ‹*infinite* (*UNIV* :: '*f set*)›
shows ‹∃*ps*. *set ps* ⊆ *X* ∧ (*ps* ⊢ *p*)›
proof (*rule ccontr*)
assume ‹∄*ps*. *set ps* ⊆ *X* ∧ (*ps* ⊢ *p*)›
then have ∗: ‹∄*ps*. *set ps* ⊆ *X* ∧ ((¬ *p*) # *ps* ⊢ ⊥)›
using *Boole* **by** *blast*

let *?S* = ‹{¬ *p*} ∪ *X*›
let *?H* = ‹*Extend ?S from-nat*›

have ‹*consistent ?S*›
using ∗ **by** (*metis consistent-def imply-Cons inconsistent-fm*)
moreover have ‹*finite* (*params ?S*)›
using *assms* **by** *simp*
ultimately have ‹*consistent ?H*› **and** ‹*maximal ?H*›
using *assms*(*3*) *consistent-Extend maximal-Extend surj-from-nat* **by** *blast*+
moreover from *this* **have** ‹*saturated ?H*›
using *saturated-Extend* **by** *fastforce*
ultimately have ‹*Hintikka ?H*›
using *assms*(*3*) *Hintikka-Extend* **by** *blast*

have ‹⟦*?H*⟧ *p*› **if** ‹*p* ∈ *?S*› **for** *p*
using *that Extend-subset Hintikka-model* ‹*Hintikka ?H*› **by** *blast*
then have ‹⟦*?H*⟧ (¬ *p*)› **and** ‹∀ *q* ∈ *X*. ⟦*?H*⟧ *q*›
by *blast*+
moreover from *this* **have** ‹⟦*?H*⟧ *p*›
using *assms*(*1*) **by** *blast*
ultimately show *False*
by *simp*
qed

theorem *completeness*:
fixes p :: ‹(*nat*, *nat*) *fm*›
assumes ‹∀(*E* :: *nat* ⇒ *nat tm*) *F G*. ⟦*E*, *F*, *G*⟧ *p*›
shows ‹⊢ *p*›
using *assms imply-completeness*[**where** *X*=‹{}›] **by** *auto*

15 Main Result

abbreviation *valid* :: ‹(*nat*, *nat*) *fm* ⇒ *bool*› **where**
‹*valid p* ≡ ∀ (*E* :: *nat* ⇒ *nat tm*) *F G*. ⟦*E*, *F*, *G*⟧ *p*›

theorem *main*: ‹*valid p* ⟷ (⊢ *p*)›
using *completeness soundness* **by** *blast*

end

References

[1] L. Henkin. The discovery of my completeness proofs. *Bulletin of Symbolic Logic*, 2(2):127–158, 1996.

[2] R. M. Smullyan. *First-Order Logic*. Springer-Verlag, 1968.